Enterprise Architektur-Muster

JULIAN BRUDER*, ABDELLAH FILALI*, and LUCA FRANKE*, Hochschule für Technik, Wirtschaft und Kultur Leipzig (HTWK Leipzig), Deutschland

Blah …

1 EINLEITUNG

Blah …

2 GRUNDLAGEN VON ENTERPRISE-ARCHITEKTUREN

Blah …

3 KLASSISCHE ENTERPRISE-ARCHITEKTUREN

Im folgenden Abschnitt werden klassische Architektur-Patterns vorgestellt. Dabei liegt der Fokus auf der monolithischen Architektur und der Layered Architektur.

3.1 Monolith

Die monolithische Architektur ist ein Softwarearchitektur-Pattern, bei dem die gesamte Funktionen in einer einzigen Anwendung zusammengefasst wird.

In der Vergangenheit wurde diese Architektur von großen Internetdiensten wie Netflix, Amazon und eBay genutzt.[Chen et al. 2017][S. 466]

Anwendungen dieser Achitektur bestehen aus eng gekoppelte Komponenten, die von einander abhängig sind. (Siehe Abb. 1), sodass sie weder eigenständig laufen noch in manchen Fällen überhaupt nicht isoliert kompiliert werden können. [Salaheddin and Ahmed 2022]

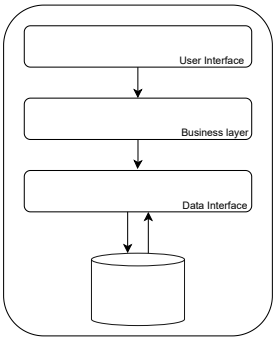

Abb. 1. Monolith Architektur

Einfachere Anwendungen, die diese Architektur nutzen, profitieren von einigen Vorteile. Sie ermöglicht eine unkomplizierte Integration von querschnittlichen Belangen wie Logging oder Sicherheit, da alle Komponenten in derselben Anwendung laufen. Ein weiterer Vorteil ist die Reduzierung des Betriebsaufwands, da nur eine zentrale Anwendung deployed und betrieben werden muss. [Kazanavičius and Mažeika 2019][S. 1]

Nachteile entstehen jedoch, wenn die Anwendung tatsächlich sehr groß und komplex wird. In diesem Fall kann die Komplexität und Unübersichtlichkeit des Code-Bases die Weiterentwicklung erschweren und Bugfixes behindern. [Chen et al. 2017][S. 466]

Große Unternehmen wie Amazon und Netflix versuchen, von Microservices zu profitieren. Allerdings ist die Migration zu Microservices nicht immer die beste Wahl. So war es beispielsweise bei Amazon Prime Video, wo die Migration rückgängig gemacht wurde und man zum Monolithen zurückkehrte. [Su and Li 2024][S. 10] In Rahmen von solchen Migrationen die modulare Monolith-Architektur wurde immer populärer. Diese kombiniert die Einfachheit der Monolithen Architektur mit den Vorteile der Microservice-Architektur, was einen Mittelweg zwischen den beiden Architekturen darstellt. [Su and Li 2024][S. 10 - S. 11] Diese Architektur wird durch die Anwendung von Domain-Driven Design auf eine traditionelle monolithische Architektur realisiert. Dadurch wird eine große Domäne in mehrere isolierte Module unterteilt, was das Aufteilen eines großen Teams in mehrere kleinere Teams ermöglicht. [Tsechelidis et al. 2023][S. 49] Außerdem besteht die Möglichkeit, die Architektur zu einem späteren Zeitpunkt in eine Microservices-Architektur zu überführen, was eine flexiblere und effizientere Alternative darstellt als eine direkte Migration [Tsechelidis et al. 2023][S. 11].

Die Module eines modularen Monolith besitzten ihre eignen Schichten mit klaren Grenzen und die Komunikation zwischen der Modulen erfolgt über APIs. Module sind lose gekoppelt und weisen eine höher Kohäsion. [Su and Li 2024]

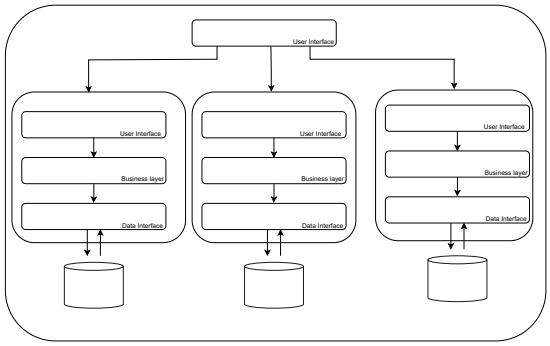

Abb. 2. Modular Monolith Architektur

Monolithen haben den Nachteil dass die Komponenten voneinander abhängig sind diesen Aspekt stellt ein Hinderniss für die agile Entwicklung, da iterative Software-Auslieferung nicht optimal geplant werden kann.

*Alle Studierenden trugen zu gleichen Teilen zu dieser Arbeit bei.

Diese Arbeit wurde im Rahmen des Mastermoduls „Software Engineering“ (Dozent: Prof. Dr. Andreas Both) an der HTWK Leipzig im Wintersemester 2024/2025 erstellt.

3.2 Layered

Das Layered Architektur-Pattern, auch bekannt als n-Tier-Architektur-Pattern, gehörte zu den am häufigsten verwendeten Architekturen. Laut [Richards 2015][S. 1] spiegelte diese Achitektur sowohl die Struktur der IT-Kommunikation als auch die organisatorische Struktur vieler Unternehmen wider, was es zur bevorzugten Wahl für die meisten geschäftlichen Anwendungsarchitekturen machte.

In dieser Architektur gibt es eine beliebige Anzahl von Schichten, die in zunehmenden Ebenen der Abstraktion und vertikal angeordnet sind (Siehe Abb. 3). Jede Schicht darf die Dienste der unmittelbar darunterliegenden Schicht nutzen.[Belle et al. 2021][S. 3]

Ein wichtige Eigenschaft der Layered Architecture ist der die Trennung der Zuständigkeiten (Englisch *Separation of concerns*). Komponenten mit unterschiedlichen Aufgaben sollten auf verschiedene Schichten verteilt werden, sodass die Komponenten einer Schicht jeweils für eine klar definierte, gemeinsame Aufgabe zuständig sind. [Tu 2023][S. 34]

Obwohl diese Architektur keine feste Anzahl an Schichten vorschreibt, bestehen die Layered Architectures meistens aus 3 Schichten [Tu 2023]:

- Presentations-Layer: Verantwortlich für die Interaktion mit dem Benutzer über die Benutzeroberfläche (UI). Sie verarbeitet Benutzeranfragen und leitet diese an die Business-Layer weiter.
- Busness-Layer: Zuständig für die Verarbeidung von Geschäftslogik und Regeln, die Weiteleitung von daten zwischen die Presentation-Layer und die Data-Layer.
- Data-Layer: Verantwortlich für die Interaktion mit Datenspeichern, wie beispielsweise Datenbanken, und die Kommunikation mit der Business-Layer, um Daten bereitzustellen oder die Ergebnisse entweder an die Presentation-Layer oder zurück an den Datenspeicher zu übermitteln. Darüber hinaus ermöglicht diese Schicht verschiedene Operationen auf den Daten, einschließlich deren Validierung sowie der Umsetzung von Sicherheitsmaßnahmen.

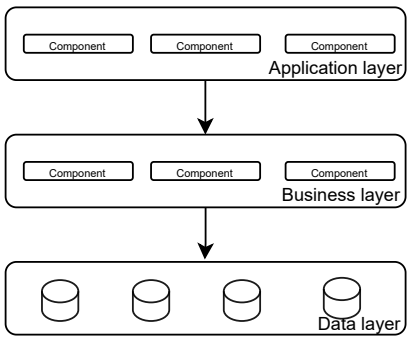

Abb. 3. Layered Architektur

Ein weiteren relevanten Konzept sind die `Open/Closed` Layers. Dieses beschreibt wie die Komunikation zwischen den Schichten erfolgt. [Thomas and Webb 2024][S. 10] Standardmäßig sind die Schichten auf Closed gesetzt. Das bedeutet, dass eine Anfrage, zwingend die direkt darunterliegende Schicht durchlaufen muss, um zur übernächsten Schicht zu gelangen [Richards 2015][S. 3].(Siehe Abb. 4) In manchen Situationen finden Architekten nützlich die Verwendung von Open Layers. Diese ermöglichen es, dass Kommunikation zwischen Schichten direkt zwischen benachbarten Schichten erfolgt, ohne die Open Layer durchlaufen zu müssen (Siehe Abb. 4). [Thomas and Webb 2024][S. 10]

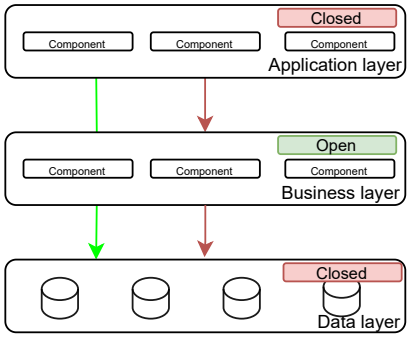

Abb. 4. Open/Close Layering

Die Implementierung dieser Architektur bietet eine Reihe von Vorteilen. Durch die Aufteilung des Systems in verschiedene Schichten wird eine unabhängige Entwicklung und Wartung der einzelnen Schichten ermöglicht [Tu 2023][S. 35]. Ein weiterer Vorteil besteht darin, dass der Zugriff auf die Dienste zwischen den Schichten über Schnittstellen (Interfaces) erfolgt. Dadurch sind Entwickler nicht gezwungen, die interne Implementierung einer Schicht zu kennen, um die bereitgestellten Dienste nutzen zu können. [Thomas and Webb 2024][S. 11] Aufgrund der losen Kopplung der Schichten ist es zudem möglich, neue Funktionalitäten hinzuzufügen oder Änderungen durch die Einführung neuer Schichten bzw. die Modifikation bestehender Schichten vorzunehmen, ohne dabei umfangreiche Anpassungen am gesamten System vornehmen zu müssen [Tu 2023][S. 35]. Allerdings führt die Nutzung dieser Architektur zu einer geringeren Performance, da für den Zugriff auf Dienste in unteren Schichten eine Kette von Anfrageweiterleitungen erforderlich ist [Thomas and Webb 2024][S. 11].

4 MODERNE ENTERPRISE-ARCHITEKTUREN

4.1 Event-Driven Architecture

Die Event-Driven Architecture wählt als Basis einen anderen Ausgangspunkt als die bisherigen Architekturmuster. Während bei letzteren Komponenten Dienste bereitstellen, welche von anderen Komponenten explizit genutzt werden, verhalten sich Dienstbereitstellende Komponenten in der Event-Driven Architecture reaktiv, werden also implizit von Dienst-konsumierenden Komponenten genutzt [Garlan and Shaw 1994]. Ein System reagiert somit asynchron auf Zustandsänderungen, also Ereignisse in diesem System [Manchana 2021]. Die in dieser Architektur minimalen Einheiten, welche Informationen einer Zustandsänderung kapseln, werden *Events* genannt. Die Idee der impliziten Behandlung von Ereignissen ist nicht neu und taucht erstmals 1994 im von Garlan und Shaw publizierten Papier *„An introduction to Software Architecture“* auf.

Betrachten wir im Folgenden die Basis-Bestandteile der Event-Driven Architecture:

- Ereignis (englisch *Event*): Kapselt Information einer Zustandsänderung eines Systems
- Produzent (englisch *Producer*): Komponente, die Event erzeugt
- Herausgeber (englisch *Publisher*): Komponente, die, von Produzenten erzeugte, Events publiziert
- Konsument (englisch *Consumer*): Reagiert auf publizierte Events
- Vermittler (englisch *Mediator*): Liegt zwischen Produzenten und Konsumenten - filtert Events und verteilt diese auf Konsumenten
- Event-Bus: Oft auch *Event-Broker* genannt - bietet die Infrastruktur für die Gesamtheit der Vermittler

Abstrakt kann ein Event als ein Vertrag zwischen Produzenten und Konsumenten am Event-Bus betrachtet werden. Der Konsument nutzt die Spezifikation des Events am Bus, der Produzent implementiert jene Spezifikation. Abbildung 5 stellt diesen Vertrag dar.

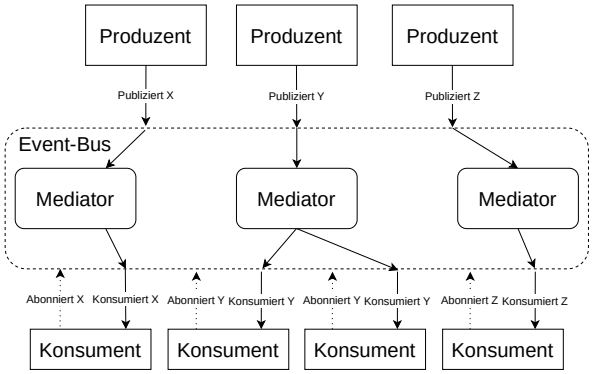

Abb. 5. Vertrag zwischen Produzenten und Konsumenten am Event-Bus

Durch den Vertrag weisen die Events am Event-Bus starke Kohäsion und somit lose Kopplung auf. Diese lose Kopplung minimiert nicht nur kaskadierende Fehler, sondern ermöglicht agilen Entwickler-Teams durch klar abgegrenzte Features einfach definierbare Iterationen - eine Menge von Events, deren Erzeugung und Konsumierung.

Weiter sind Events oft nah an dem, was Ereignisse in realen Prozessen sind, also domain-driven. Gebündelt ermöglichen obige Punkte die kontinuierliche Auslieferung von Software in kurzen Intervallen.

Außerdem garantiert die asynchrone Behandlung von Ereignissen zusammen mit der loosen Kopplung maximale Skalierung. Daher sind Event-Driven Architekturen besonders für datenintensive Echtzeit-Anwendungen wie IoT (Internet of Things) und Analytics geeignet [Siddiqui et al. 2023].

Die Agilität der Architektur kann weiter erhöht werden, indem der event-basierte Aspekt mit weiteren agilen Strukturen wie Microservices oder cloud-nativen Serverless-Functions kombiniert wird. Die damit einhergehende Komplexität stellt teilweise hohe Anforderungen an die Entwickler. Aufgrund der Asynchronität der Behandlung von Ereignissen ist die Testung des Systems meist schwer und die Fehlerbehandlung essentiell. Mögliche Problemquellen schließen dabei unter anderem Event-Verlust, erhöhte Latenz und Inkonsistenz ein. Die hohen Anforderungen an die Entwickler verlangen viel Vertrauen in jene, einer der zentralen Punkte des agilen Manifests [Michl 2018]. Insgesamt weist die Event-Driven Architecture also eine sehr hohe Agilität auf und ist damit besonders für moderne Software und ihre stetig wechselnden Anforderungen geeignet.

5 FALLSTUDIEN UND PRAXISBEISPIELE

Blah …

6 DISKUSSION

7 ZUSAMMENFASSUNG UND AUSBLICK

LITERATUR

Alvine B. Belle, Ghizlane El Boussaidi, Timothy C. Lethbridge, Segla Kpodjedo, Hafedh Mili, and Andres Paz. 2021. Systematically reviewing the layered architectural pattern principles and their use to reconstruct software architectures. arXiv:2112.01644 [cs.SE] https://arxiv.org/abs/2112.01644

Rui Chen, Shanshan Li, and Zheng Li. 2017. From Monolith to Microservices: A Dataflow-Driven Approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 466–475. https://doi.org/10.1109/APSEC.2017.53

David Garlan and Mary Shaw. 1994. *An Introduction to Software Architecture.* Technical Report CMU/SEI-94-TR-021. https://insights.sei.cmu.edu/library/an-introduction-to-software-architecture/ Accessed: 2025-Jan-2.

Justas Kazanavičius and Dalius Mažeika. 2019. Migrating Legacy Software to Microservices Architecture. In *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. 1–5. https://doi.org/10.1109/eStream.2019.8732170

Ramakrishna Manchana. 2021. Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries. *International Journal of Science and Research (IJSR)* 10 (01 2021), 1706–1716. https://doi.org/10.21275/SR24820051042

Thomas Michl. 2018. *Das agile Manifest – eine Einführung.* Springer Berlin Heidelberg, Berlin, Heidelberg, 3–13. https://doi.org/10.1007/978-3-662-57699-1_1

Mark Richards. 2015. *Software architecture patterns.* Vol. 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ….

Nada Salaheddin and Nuredin Ahmed. 2022. MICROSERVICES VS. MONOLITHIC ARCHITECTURE [THE DIFFERENTIAL STRUCTURE BETWEEN TWO ARCHITECTURES]. *MINAR International Journal of Applied Sciences and Technology* 4 (10 2022), 484–490. https://doi.org/10.47832/2717-8234.12.47

Hassaan Siddiqui, Ferhat Khendek, and Maria Toeroe. 2023. Microservices based architectures for IoT systems - State-of-the-art review. *Internet of Things* 23 (2023), 100854. https://doi.org/10.1016/j.iot.2023.100854

Ruoyu Su and Xiaozhou Li. 2024. Modular Monolith: Is This the Trend in Software Architecture?. In *Proceedings of the 1st International Workshop on New Trends in Software Architecture* (Lisbon, Portugal) *(SATrends '24)*. Association for Computing Machinery, New York, NY, USA, 10–13. https://doi.org/10.1145/3643657.3643911

Richard Thomas and Brae Webb. 2024. Layered architecture. (2 2024). https://csse6400.uqcloud.net/handouts/layered.pdf

Michail Tsechelidis, Nikolaos Nikolaidis, Theodore Maikantis, and Apostolos Ampatzoglou. 2023. Modular Monoliths the way to Standardization. In *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum* (Ludwigsburg, Germany) *(eSAAM '23)*. Association for Computing Machinery, New York, NY, USA, 49–52. https://doi.org/10.1145/3624486.3624506

Zhenan Tu. 2023. Research on the Application of Layered Architecture in Computer Software Development. *Journal of Computing and Electronic Information Management* 11 (11 2023), 34–38. https://doi.org/ojs/index.php/jceim/article/view/14398

A ANHANG 1

A.1 Übungsaufgaben

Blah …

B ANHANG 2

Blah …